Genetic Line Search

S. Lozano[1], J.J. Domínguez[2], F. Guerrero[1], and K. Smith[3]

[1]Escuela Superior de Ingenieros, Camino de los Descubrimientos, s/n, 41092 Sevilla, Spain
slozano@cica.es, fergue@esi.us.es
[2]Dpto. Lenguajes y Sistemas Informáticos - University of Cadiz, Spain
juanjose.dominguez@uca.es
[3]School of Business Systems, Monash University, Clayton, Victoria 3800, Australia
kate.smith@infotech.monash.edu.au

Abstract. All unconstrained and many constrained optimization problems involve line searches, i.e. minimizing the value of a certain function along a properly chosen direction. There are several methods for performing such one-dimensional optimization but all of them require that the function be unimodal along the search interval. That may force small step sizes and in any case convergence to the closest local optimum. For multimodal functions a line search along any direction is likely to have multiple valleys. We propose using a Genetic Line Search with scalar-coded individuals, convex linear combination crossover and niche formation. Computational experiences show that this approach is more robust with respect to the starting point and that a fewer number of line searches is usually required.

1. Introduction

Unconstrained optimization deals with the problem of minimizing (or maximizing) a function in the absence of any restrictions, i.e.

$$\underset{\bar{x}\in\Re^n}{Min} \quad f(\bar{x}) \ . \tag{1}$$

Most solution methods consist in repeating a basic two-phase process: determining a search direction $\bar{d}^k$ and a step length α_k^* such that

$$\bar{x}^{k+1} \quad \leftarrow \quad \bar{x}^k + \alpha_k^* \, \bar{d}^k \ . \tag{2}$$

How to carry out such iterative process depends on the differentiability or not of function *f*. The Cyclic Coordinate Method, the Method of Hooke and Jeeves and the Rosenbrock Method determine the search direction without need of derivatives information [1]. On the contrary, the Gradient Method, Newton's Method, Quasi-Newton and Conjugate Gradient Methods all require that function *f* is at least once (more often twice) continuously differentiable. All these methods require performing a number of line searches. Thus, line searches can be considered the backbone of all these unconstrained optimization approaches. It can even be argued that they also play a basic role in constrained optimization since many algorithms solve a constrained problem through a sequence of unconstrained problems via Lagrange relaxation or penalty and barrier functions.

A line search can be exact or inexact. The first case is equivalent to a one-dimensional optimization problem

$$\underset{\alpha_k \in (0, \alpha_k^{max})}{Min} \quad F(\alpha_k) \equiv f(\bar{x} + \alpha_k \bar{d}^k) \ . \tag{3}$$

where the search interval *$[0, \forall_k^{max}]$* depends on the maximum stepsize, which guarantees a unique minimum in the interval of uncertainty. Most exact line search methods use such strict unimodality requirement to iteratively reduce the interval of uncertainty by excluding portions of it that do not contain the minimum. They differ in whether or not they use derivative information. Line search without using derivatives include Dichotomous Search, Fibonacci Search and Golden Section Method [1]. Line search using derivatives include Bisection Search and Newton's Method. Polynomial (quadratic or cubic) Interpolation Methods may or may not use derivative information [1].

Very often, in practice, the expense of excessive function and gradient evaluations forces a finish of the line search before finding the true minimum of *$F(\forall_k)$* in (3). Since that may impair the convergence of the overall algorithm that employs the line search, different conditions may be imposed for terminating the search provided that a significant reduction in the objective function *f* has been achieved [2].

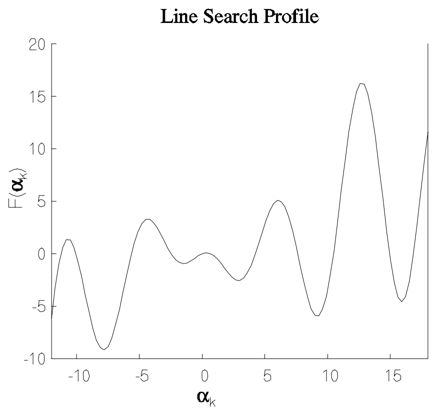

Fig. 1. Typical multi-modal function profile along a search direction

The requirement that the search interval has a unique minimum implies that only the first valley along the search direction is explored and that the overall optimization algorithm cannot converge to a local minimum other than the one at the bottom of the valley in which the starting point $\bar{x}$ lies. For multi-modal functions, many valleys are likely to happen along any search direction as shown in Figure 1. Some of those valleys may be deeper than the first one. It would be sensible to move to a deeper valley since that would give a bigger reduction in *f*. In other words, freeing the line search from restricting itself to the neighborhood of $\bar{x}$ allows for jumping across valleys, thus accelerating the optimization process.

To implement such a look-further-ahead strategy a robust and efficient procedure must be used to solve a one-dimensional, multi-modal optimization problem. Genetic Algorithms (GA) are well-suited for this task since they do not require the function to be differentiable (not even continuous) and can be parallel-processed. In [3] combining GA and local optimization of a continuous function is suggested. It is proposed the use of the conventional methods for unconstrained optimization that we have mentioned above, starting with the solution obtained by the GA or, better yet, augmenting the GA with a problem-specific hill-climbing operator yielding a hybrid GA. Our approach is completely different. We embed the GA in the overall optimization process replacing traditional line search methods.

The rest of this paper has the following structure. Section 2 describes the application of GA to the line search task while Section 3 show some computational experiences from which, in Section 4, conclusions are drawn.

2. Genetic Line Search

The Genetic Line Search (GLS) algorithm we propose has the following features:

- Individuals are scalars, coded as floating-point real numbers. Let $\forall_{knt}$ the n-th individual of the population after t iterations in the k-th line search.
- Two operators are used, one for crossover and another for mutation. The crossover operator is a convex linear combination operator. The mutation operator consists in generating a random gaussian perturbation around the selected individual.

$$\alpha_{knt} \leftarrow \alpha_{knt} + N(0,\sigma) \; . \tag{4}$$

  The magnitude of the perturbation is controlled through the standard deviation σ, which is kept constant at a value proportional to one minus the probability of mutation. The rationale is that the more frequent mutations are performed, the smaller their magnitude and vice versa.
- Fitness is defined as minus the objective function for a minimization problem. It would be equal to the objective function in case of a maximization problem. The population is kept always ordered according to fitness. In order to keep fitness values positive adding a constant may be required

$$F(\alpha_{knt}) \leftarrow F(\alpha_{knt}) + \left| \underset{j \in Population(t)}{Min} F(\alpha_{kjt}) \right| + 1 \; . \tag{5}$$

- A steady state GA has been implemented. In each iteration either mutation (with probability p_m) or crossover (with probability $p_c=1-p_m$) is applied. Mutation requires just one individual and produces another one. Crossover requires two parents and generates two off-springs of which the least fit is discarded. Therefore, the population is renewed at a rate of one individual per iteration.
- The individual to be deleted from the population is selected independently of the individuals to reproduce. Individuals to reproduce through mutation or crossover are selected with uniform probability, i.e. selection for reproduction is not fitness-

biased. On the contrary, selection for deletion is rank-based with a (non-linear) geometric probability distribution [4].

- In order to promote diversification, a sharing function scheme which encourages niche formation is used [3]. It modifies the fitness landscape penalizing the accumulation of individuals near one another. When several individuals are very similar to each other they mutually reduce their fitness. There is a distance threshold, called the niche size, which sets the minimum distance of separation of two individuals for them not to interfere each other. Instead of directly fixing that parameter, the user is asked for a desired number of solutions per niche. The population size divided by the number of solutions per niche gives the expected number of niches and from that the estimated niche size is derived.
- The parameters that have been introduced so far are: population size, crossover and mutation probabilities and number of solutions per niche. Another parameter to be fixed is the number of iterations after which the GLS stops, returning the best individual in the final population. Since the selection for deletion is population-elitist there is no need to distinguish between on-line and off-line performance. In case a different GA were used, off-line performance should be monitored.

There is yet another parameter and an important one for that matter. It is the problem scale *S* which will be used for the different line searches. This parameter specifies whether the magnitudes of the decision variables are of order unity or of the order of thousands or millions. This helps in computing the maximum and minimum stepsizes in each line search. But before we express how stepsizes are bounded, some considerations are in order:

- Even in the case of unconstrained optimization it is always possible for real-world problems to set upper and lower limits on most if not all variables. Let l_i and u_i respectively the lower and upper bounds on variable *i*.
- Limits on the decision variables translate, through the components d_i^k of the search direction, into limits on the allowed stepsize for line search *k*.
- The stepsize scale in line search *k* is related to the problem scale *S* through the norm of the search direction $\left\|\bar{d}^k\right\|$.Thus, if the search direction is not normalized, problem scale must be adjusted to give the proper stepsize scale.
- Besides the previous limits on the stepsize, the user may impose an overall maximum stepsize $\forall^{max}$ valid for all line searches.
- Conventional line search techniques do not explore negative values for the stepsize since the search direction is normally chosen so as to decrease the objective function locally in that direction. However, our approach is not restricted to the neighborhood of the line search origin and although it can be expected that small negative values increase the objective function value, further away along the negative axis there may be deep valleys worth exploring. Therefore, our approach sets a negative lower bound on the step size. In the end we have

$$\alpha_k^{min} \le \alpha_k \le \alpha_k^{max} \ . \tag{6}$$

where

$$0 \le \alpha_k^{max} \le Min \left\{ \underset{i:d_i^k > 0}{Min} \frac{u_i - x_i^k}{d_i^k}, \underset{i:d_i^k < 0}{Min} \frac{l_i - x_i^k}{d_i^k}, \frac{S}{\left\| \bar{d}^k \right\|}, \alpha^{max} \right\} . \tag{7}$$

$$0 \ge \alpha_k^{min} \le Max \left\{ \underset{i:d_i^k < 0}{Max} \frac{u_i - x_i^k}{d_i^k}, \underset{i:d_i^k > 0}{Max} \frac{l_i - x_i^k}{d_i^k}, -\frac{S}{\left\| \bar{d}^k \right\|}, -\alpha^{max} \right\} . \tag{8}$$

Whenever a mutation causes an individual to step out of this interval, it is repaired by setting its value equal to the closest feasible value. Since the convex crossover operator guarantees that if the two parents fall inside a closed interval, so does every off-spring generated in such a way, no reparation is ever required after applying crossover.

The stepsize bounding mechanism described above may seem a bit complicated but a proper determination of the search interval prior to starting each line search can significantly improve the overall performance of the optimization algorithm. Too narrow a search interval is a waste from the GLS point of view. On the other side, too wide a search interval may cause the GLS to be stretched too thin. Note, finally, that GLS explores a $[\forall_k^{min}, \forall_k^{max}]$ search interval that resembles but differs from the interval $[0, \forall_k^{max}]$ in equation (1). Not only negative values are allowed but also the interval scale is larger, usually involving multiple valleys in which to move into. This feature is crucial if better performance than conventional methods is aimed.

A pseudocode description of the GLS algorithm follows:

```
procedure GLS
begin
    t 7 1
    initialize Population(1)
    evaluate adjusted fitness and order Population(1)
    while t < number-of-iterations-allowed do
    begin
        t• 7 +1
        select operator (crossover or mutation)
        perform selection for reproduction
        apply operator
        perform selection for deletion
        insert new individual
        readjust fitness and reorder Population(t)
    end
end
```

As it has been said before, GLS is embedded in an overall optimization algorithm whose pseudocode description is:

```
procedure optimization
begin
```

```
        input starting point $\bar{x}^1$
        input problem scale $S$
        input (if any) bounds $l_i$ and $u_i$
        input (if any) maximum stepsize $\alpha^{max}$
        k• 7 1
        select search direction $\bar{d}^k$
        perform GLS from $\bar{x}^k$ along $\bar{d}^k$ obtaining $\bar{x}^{k+1}$
        while significant-reduction-obtained do
        begin
            k  7 k+1
            select search direction $\bar{d}^k$
            perform GLS from $\bar{x}^k$ along $\bar{d}^k$ obtaining $\bar{x}^{k+1}$
        end
    end
```

Note that the proposed termination condition is based on the amount of reduction in the objective function obtained in the last line search, i.e.

$$\frac{\left|f(\bar{x}^k)-f(\bar{x}^{k+1})\right|}{\left|f(\bar{x}^k)\right|} \quad . \tag{9}$$

3. Computational Experiences

There exists in the literature a set of standard problems [5] to help compare the relative performance of different line search methods. A thorough study to benchmark GLS will be the subject of another paper. What we report here are the results obtained with the proposed approach for just two of the problems tested. Results of using GLS to train feed-forward Neural Networks have been reported in [6].

The first test is the trigonometric function (TRIG) which in the interval under consideration $[0,100]^n$ has many local minima and a global minimum in the vicinity of the point (100,100,...,100). The second one is the Rastrigin function (RAST) which in the interval under consideration $[-50,50]^n$ has many local minima and a global minimum at (0,0,...,0).

Figure 2 shows the successive line searches performed by GLS and a standard Golden Section algorithm [1] for RAST function starting at (-49, 47) and using Cyclic Coordinate Method for selecting the direction of descent. While GOLDEN requires multiple small-step line searches and gets stuck at a near-by local minimum, GLS require just two line searches to reach the global minimum.

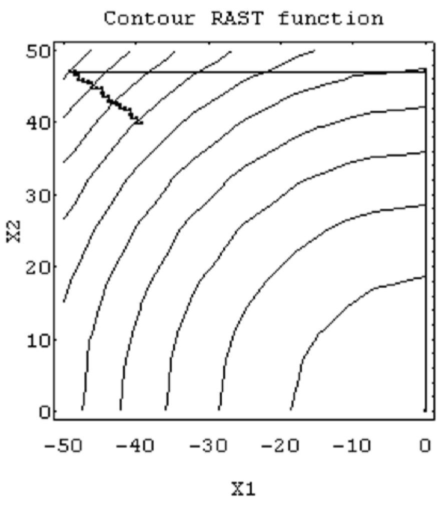

Fig. 2. Succesive line searches starting at (-49,47)

Figures 3 and 4 show the results for minimizing RAST function starting from 100 random points. GLS always requires more computing time but, independently of the starting point, almost always reaches the global minimum. Golden, on the other hand, is faster but it converges to the optimum only when the starting point is close.

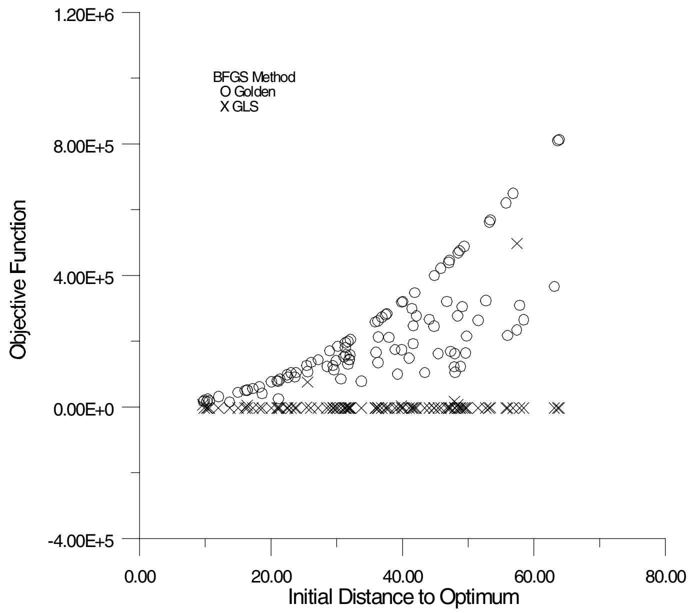

Fig. 3. Initial Distance to Optimum vs final RAST Obj. Funct. for 100 random starting points

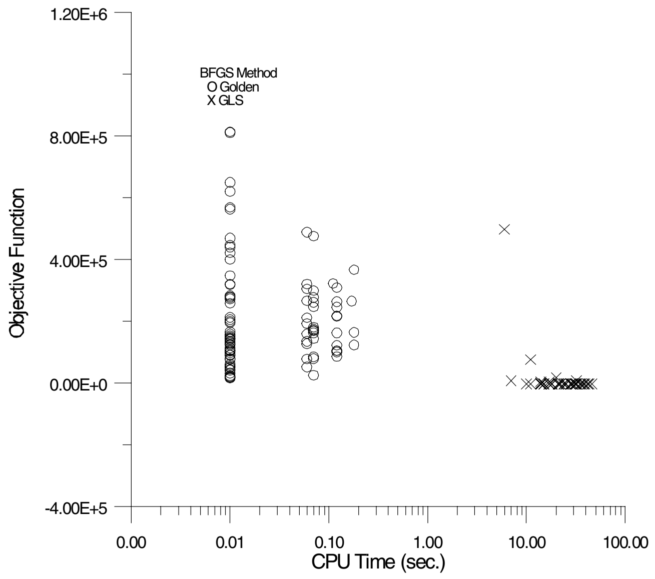

Fig. 4. CPU Time vs final RAST Objective Function for 100 random starting points

The results shown so far correspond to the two-variables case. To test the scalability of the methods, we considered an increasing number of variables: 5, 10, 20, 25, 50, 75 and 100. Figure 5 shows how both Golden and GLS find lower values of TRIG Objective Function as the number of variables increases although GLS always finds better values than Golden. The gap seems to widen with the number of variables.

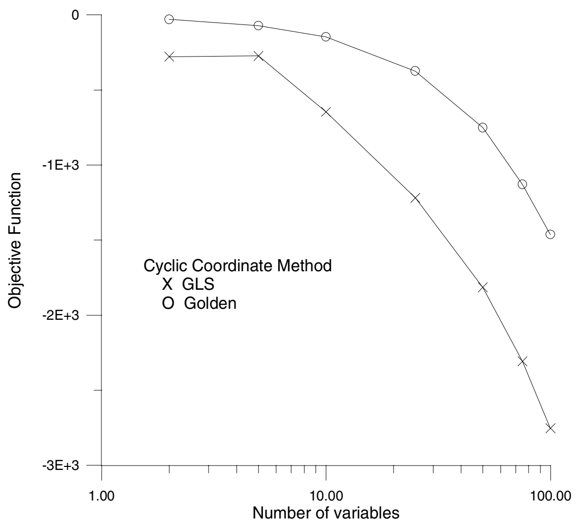

Fig. 5. TRIG Objective Function vs number of variables

Figure 6 shows the effect of the number of variables on CPU Time. Both Golden and GLS require increasing CPU Time and although Golden is always faster than GLS, its requirements seem to increase at a faster rate than those of GLS.

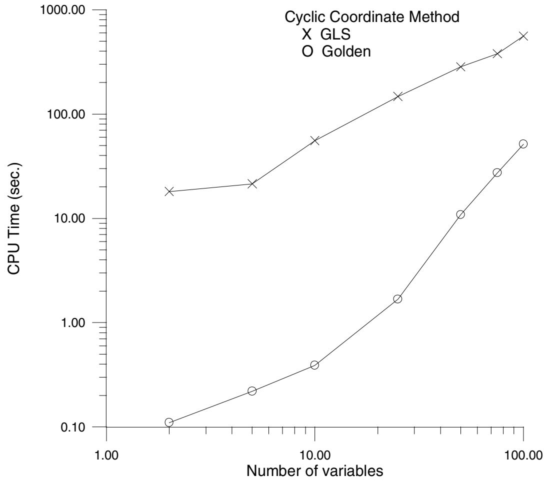

Fig. 6. CPU Time vs number of variables

4. Summary

This paper deals with a new approach to unconstrained optimization of multi-modal functions, consisting on performing successive line searches using a Genetic Algorithm. Genetic Line Search explores larger intervals along the search direction which translate into larger step sizes. This look-further-ahead strategy overcomes the local nature of traditional line search methods allowing the algorithm to escape from local optima. The experiments carried out show that GLS requires more computing time for each line search but requires a smaller number of them, is more robust with respect the starting point and generally obtains better values of the objective function.

References

1. Bazaraa, M.S., H.D. Sherali and C.M. Shetty, *Nonlinear Programming. Theory and Algorithms*, Wiley, Singapore, 2nd edition (1993)
2. Fletcher, R., *Practical Methods of Optimization*, Wiley, Chichester, 2nd edition (1987)
3. Goldberg, D.E., *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley, Reading (1989)
4. Michalewicz, Z., *Genetic Algorithms + Data Structures = Evolution Programs*, Springer-Verlag, New York (1992)
5. More, J.J., B.S. Garbow and K.E. Hillstrom, "Testing Unconstrained Optimization Software", *ACM Transactions on Mathematical Software*, vol. 7, 1 (1981) pp. 17-41
6. Lozano, S., J.J. Dominguez, F. Guerrero, L. Onieva and J. Larrañeta, "Training Feedforward Neural Networks Using a Genetic Line Search", in *Smart Engineering System: Neural Networks, Fuzzy Logic, Data Mining and Evolutionary Programming*, C.H. Dagli, M. Akay, O. Ersoy, B.R. Fernández and A. Smith (eds.), ASME Press, New York (1997) pp. 119-124